In [1]:

```
import pandas as pd
```

In [2]:

```
df = pd.read_csv('diabetes.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
2	183	64	0	0	23.3	0.672	32	1
3	89	66	23	94	28.1	0.167	21	0
4	137	40	35	168	43.1	2.288	33	1

In [4]:

```
df.tail()
```

Out[4]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
768	0	0	0	0	0.0	0.0	0	0
769	0	0	0	0	0.0	0.0	0	0
770	0	0	0	0	0.0	0.0	0	0
771	0	0	0	0	0.0	0.0	0	0
772	0	0	0	0	0.0	0.0	0	0

In [5]:

```
df.shape
```

Out[5]:

```
(773, 8)
```

In [6]:

```
df.columns
```

Out[6]:

```
Index(['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
       'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [7]:

```
df.duplicated().sum()
```

Out[7]:

```
4
```

In [8]:

```
df.isnull().sum()
```

Out[8]:

```
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 773 entries, 0 to 772
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Glucose                   773 non-null    int64  
 1   BloodPressure             773 non-null    int64  
 2   SkinThickness             773 non-null    int64  
 3   Insulin                   773 non-null    int64  
 4   BMI                       773 non-null    float64
 5   DiabetesPedigreeFunction  773 non-null    float64
 6   Age                       773 non-null    int64  
 7   Outcome                   773 non-null    int64  
dtypes: float64(2), int64(6)
memory usage: 48.4 KB
```

In [10]:

```
df.describe()
```

Out[10]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
count	773.000000	773.000000	773.000000	773.000000	773.000000	773.000000
mean	120.112549	68.658473	20.403622	79.283312	31.785640	0.468824
std	33.311787	20.073629	15.985586	115.048418	8.267017	0.332416
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	99.000000	62.000000	0.000000	0.000000	27.200000	0.240000
50%	117.000000	72.000000	23.000000	23.000000	32.000000	0.370000
75%	140.000000	80.000000	32.000000	126.000000	36.500000	0.624000
max	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000

In [11]:

```
df.nunique()
```

Out[11]:

```
Glucose                     136
BloodPressure                47
SkinThickness                51
Insulin                     186
BMI                         248
DiabetesPedigreeFunction    518
Age                          53
Outcome                       2
dtype: int64
```

In [12]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [13]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [14]:

```
df['Outcome'].unique()
```

Out[14]:

```
array([1, 0], dtype=int64)
```

In [15]:

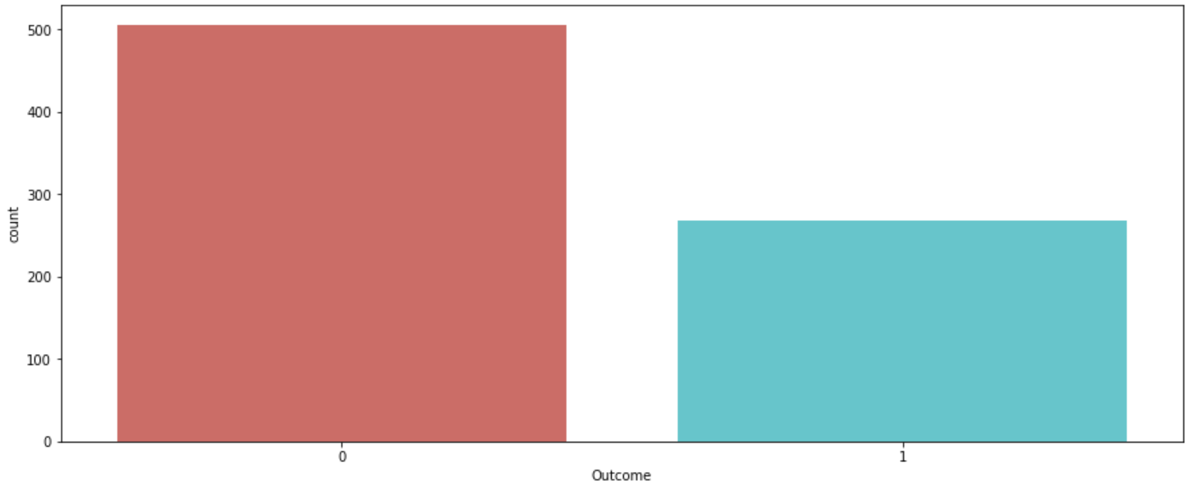
```
df['Outcome'].value_counts()
```

Out[15]:

```
0    505
1    268
Name: Outcome, dtype: int64
```

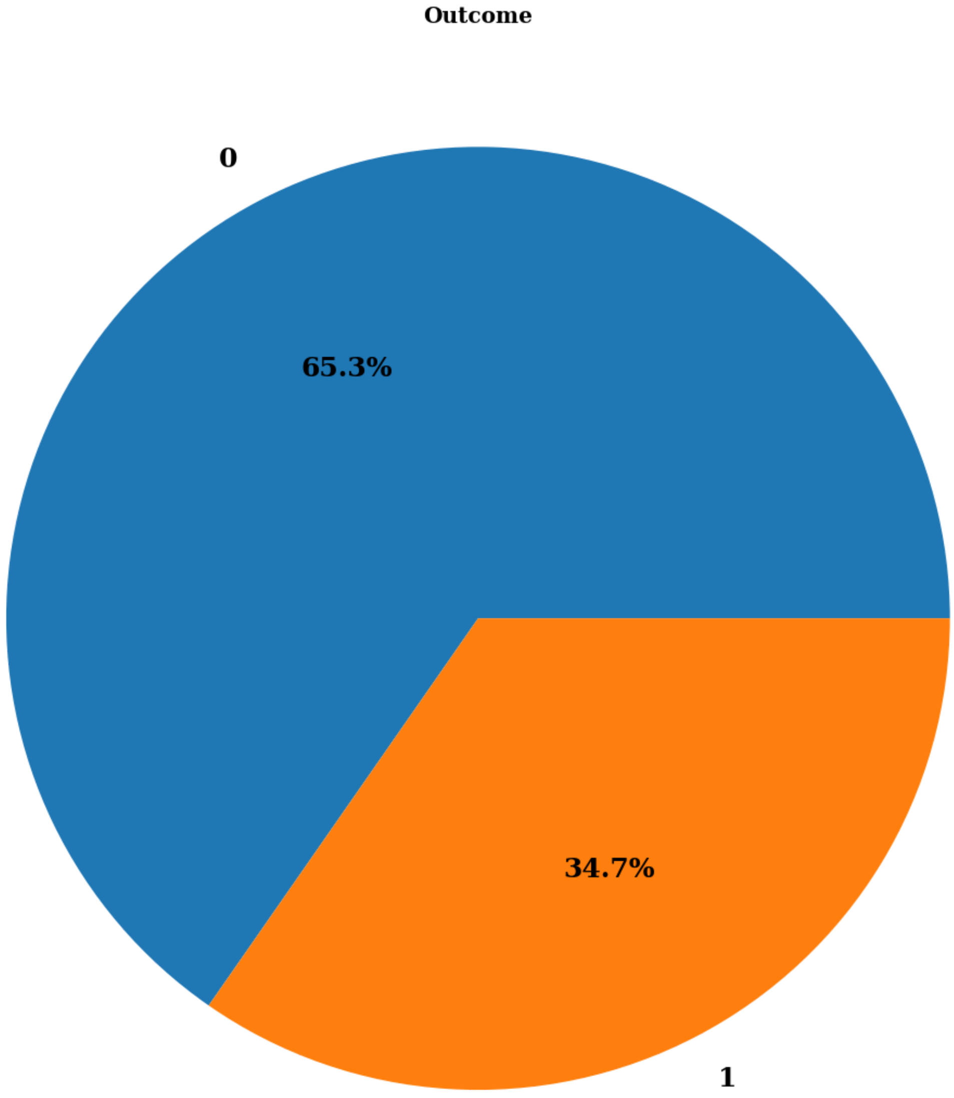
In [16]:

```
plt.figure(figsize=(15,6))
sns.countplot(df['Outcome'], data = df, palette = 'hls')
plt.show()
```



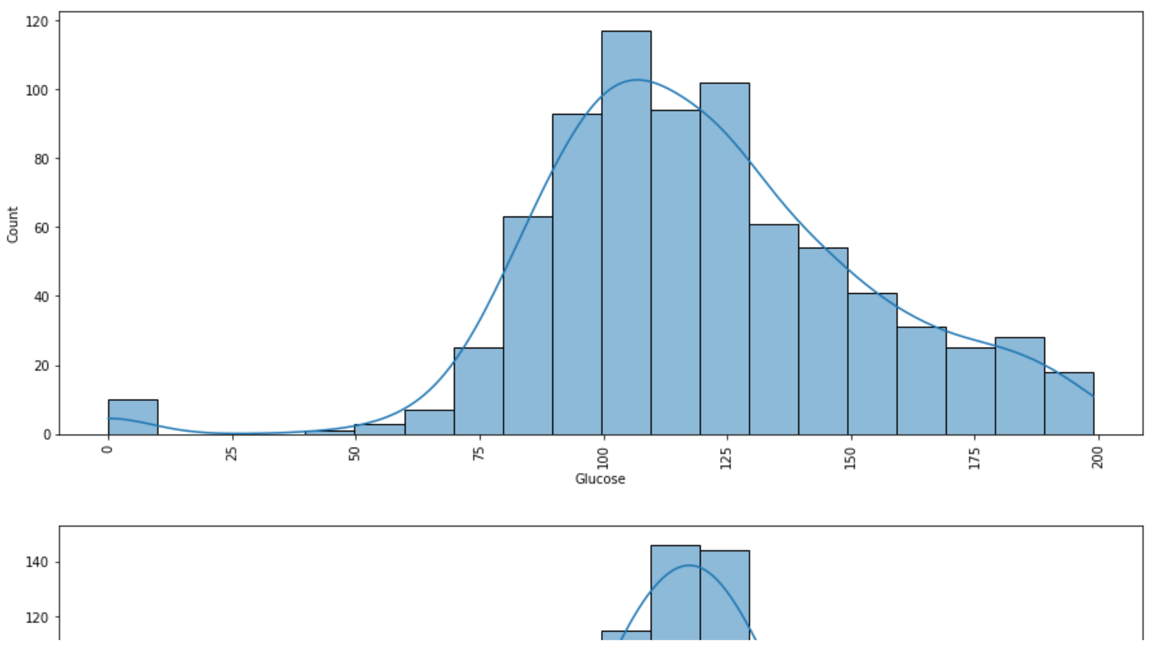
In [17]:

```
plt.figure(figsize=(30,20))
plt.pie(df['Outcome'].value_counts(), labels=df['Outcome'].value_counts().index, autopct='%1.1
                                          'color': 'black',
                                          'weight': 'bold',
                                          'family': 'serif' })
hfont = {'fontname':'serif', 'weight': 'bold'}
plt.title('Outcome', size=20, **hfont)
plt.show()
```



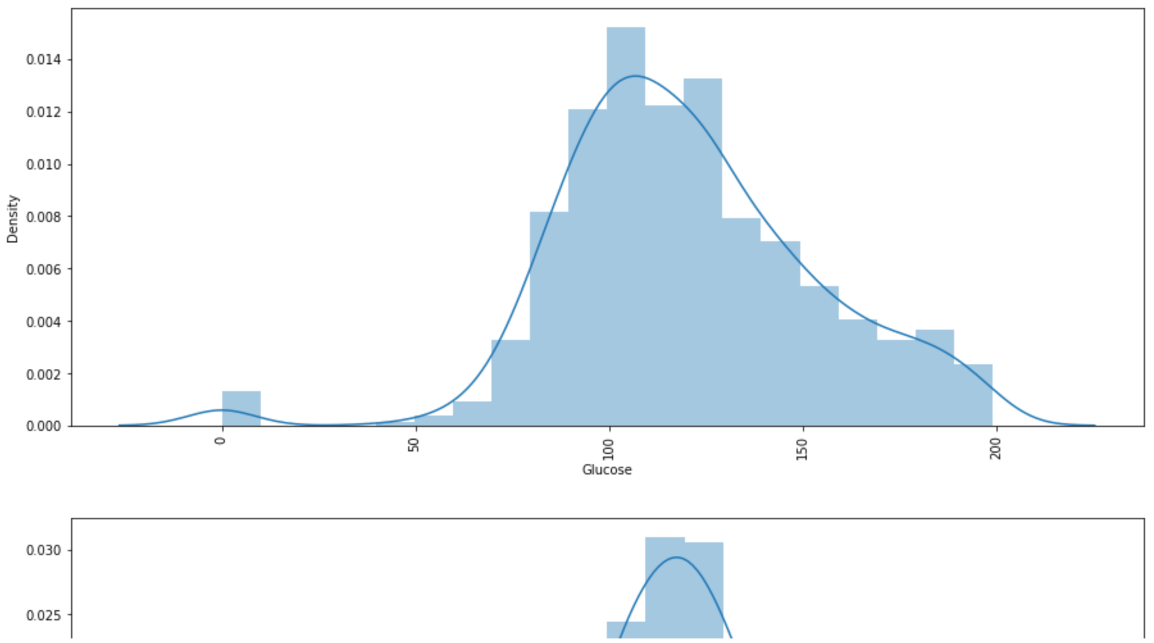
In [18]:

```
for i in df.columns:
    plt.figure(figsize=(15,6))
    sns.histplot(df[i], kde = True, bins = 20, palette = 'hls')
    plt.xticks(rotation = 90)
    plt.show()
```



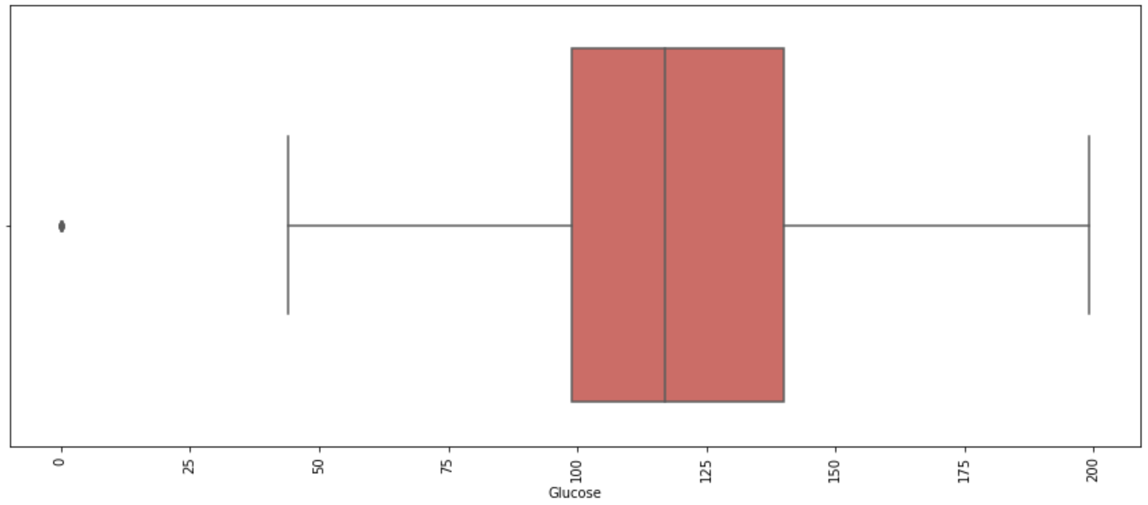
In [19]:

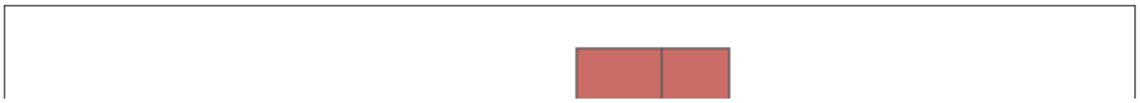
```
for i in df.columns:
    plt.figure(figsize=(15,6))
    sns.distplot(df[i], kde = True, bins = 20)
    plt.xticks(rotation = 90)
    plt.show()
```



In [22]:

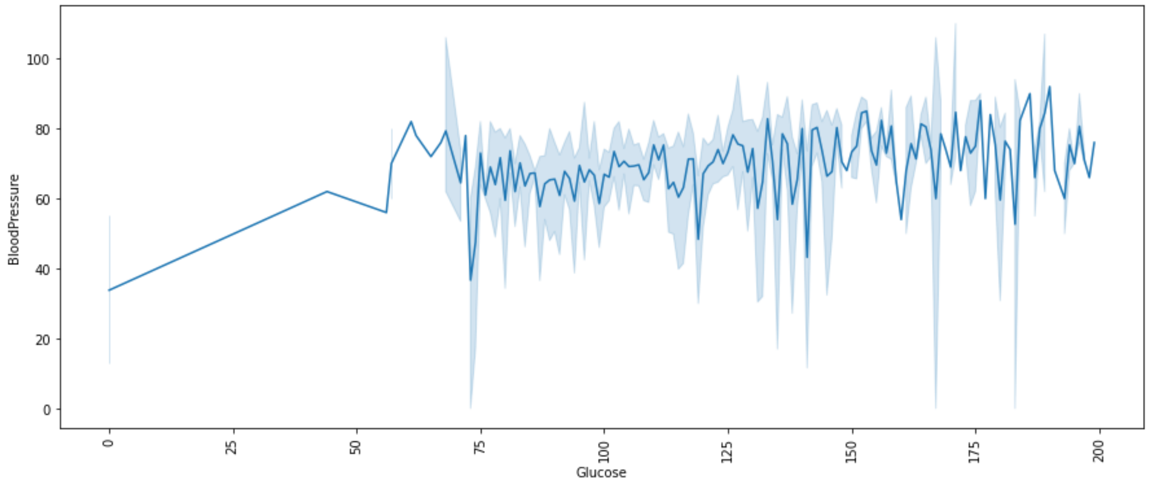
```
for i in df.columns:
    plt.figure(figsize=(15,6))
    sns.boxplot(df[i], data = df, palette = 'hls')
    plt.xticks(rotation = 90)
    plt.show()
```





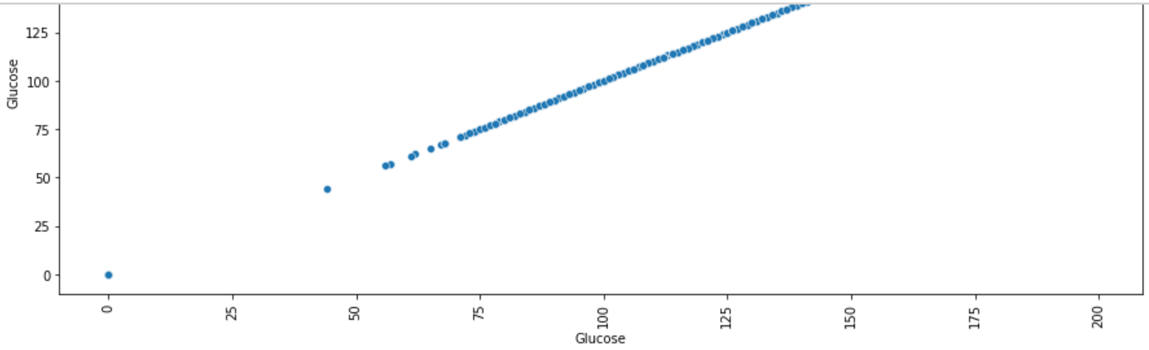
In [24]:

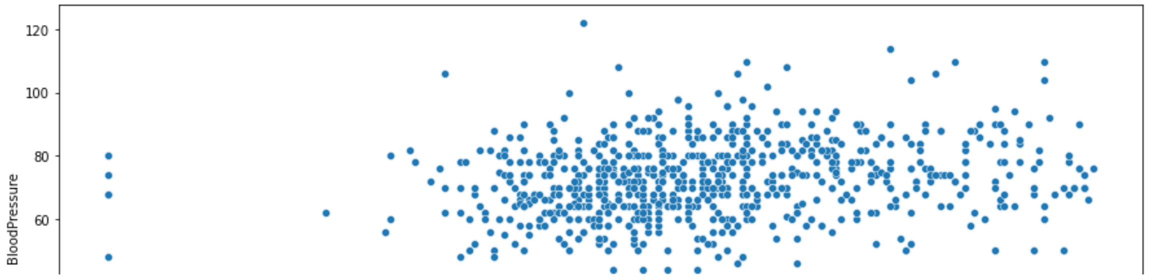
```
for i in df.columns:
    for j in df.columns:
        plt.figure(figsize=(15,6))
        sns.lineplot(x = df[i], y = df[j], data = df, palette = 'hls')
        plt.xticks(rotation = 90)
        plt.show()
```



In [25]:

```
for i in df.columns:
    for j in df.columns:
        plt.figure(figsize=(15,6))
        sns.scatterplot(x = df[i], y = df[j], data = df, palette = 'hls')
        plt.xticks(rotation = 90)
        plt.show()
```



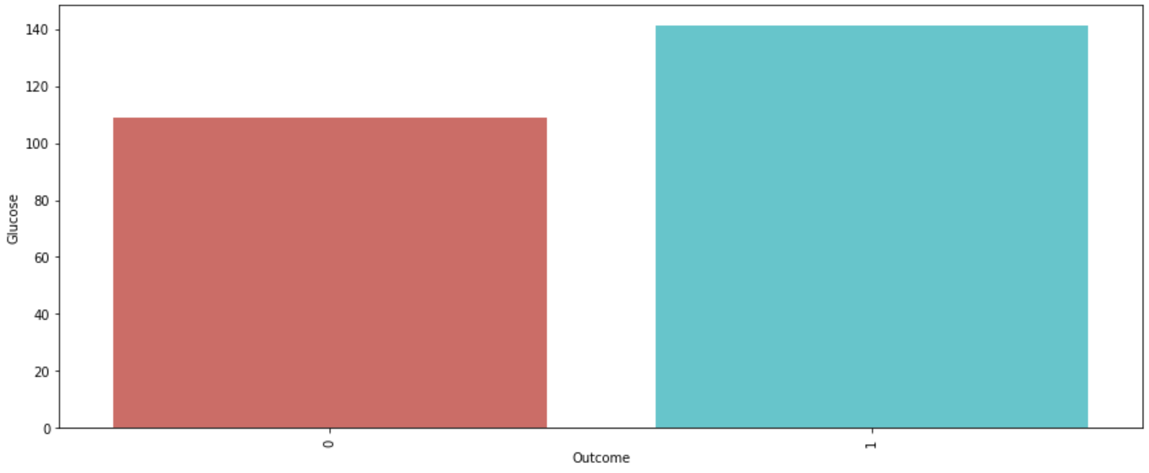


In [26]:

```
df_new = df.drop(['Outcome'], axis = 1)
```

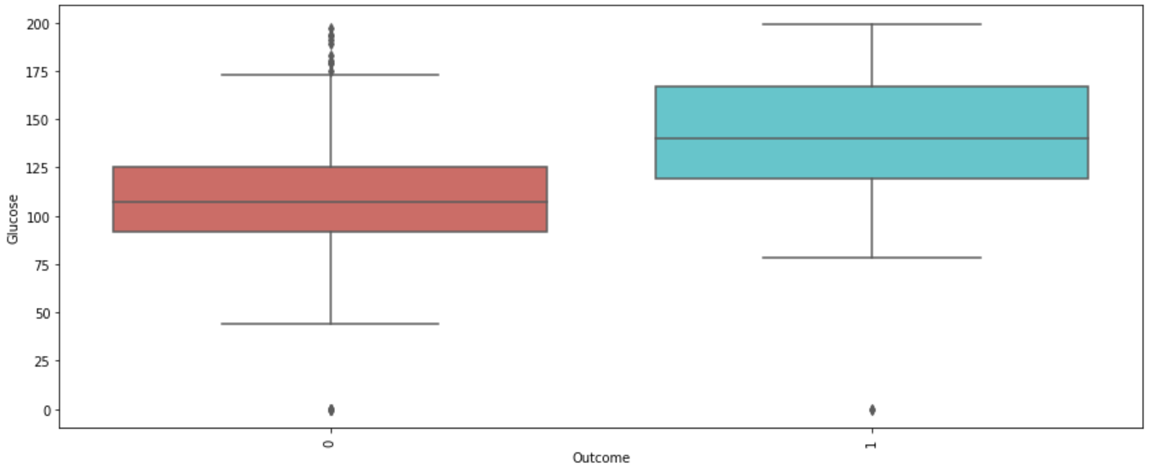
In [28]:

```
for i in df_new.columns:
    plt.figure(figsize=(15,6))
    sns.barplot(x = df['Outcome'], y = df_new[i], data = df, ci = None, palette = 'hls')
    plt.xticks(rotation = 90)
    plt.show()
```



In [30]:

```
for i in df_new.columns:
    plt.figure(figsize=(15,6))
    sns.boxplot(x = df['Outcome'], y = df_new[i], data = df, palette = 'hls')
    plt.xticks(rotation = 90)
    plt.show()
```



In [31]:

```
import numpy as np
```

In [32]:

```
df_corr = df.corr()
```

In [33]:

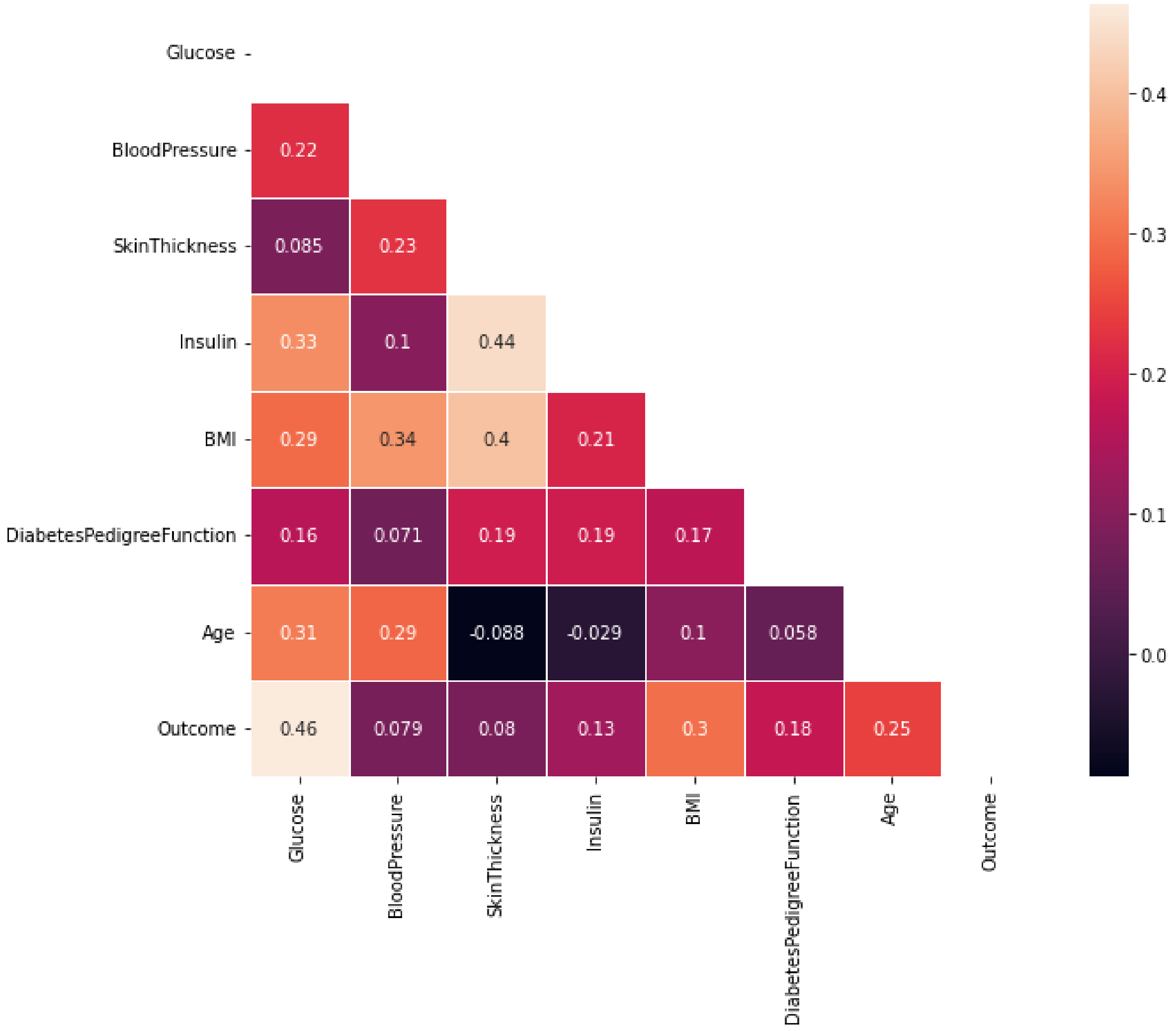
```
df_corr
```

Out[33]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPed
Glucose	1.000000	0.220699	0.084554	0.332712	0.291421	
BloodPressure	0.220699	1.000000	0.226704	0.100708	0.343193	
SkinThickness	0.084554	0.226704	1.000000	0.439518	0.403183	
Insulin	0.332712	0.100708	0.439518	1.000000	0.205065	
BMI	0.291421	0.343193	0.403183	0.205065	1.000000	
DiabetesPedigreeFunction	0.163684	0.070848	0.193493	0.189918	0.168178	
Age	0.310394	0.285733	-0.087681	-0.028708	0.102450	
Outcome	0.462712	0.078662	0.080283	0.133391	0.296000	

In [34]:

```
plt.figure(figsize=(10, 8))
matrix = np.triu(df_corr)
sns.heatmap(df_corr, annot=True, linewidth=.8, mask=matrix, cmap="rocket");
plt.show()
```



In [35]:

```
X = df.drop(['Outcome'], axis = 1)
y = df['Outcome']
```

In [36]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [37]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
```

In [38]:

```
from sklearn.linear_model import LogisticRegression
```

In [39]:

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

Out[39]:

```
▾ LogisticRegression
LogisticRegression()
```

In [40]:

```
y_pred = model.predict(X_test)
```

In [41]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_
```

In [42]:

```
# calculate accuracy, precision, recall, and f1-score
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1))
```

```
Accuracy: 0.80
Precision: 0.78
Recall: 0.56
F1-score: 0.65
```

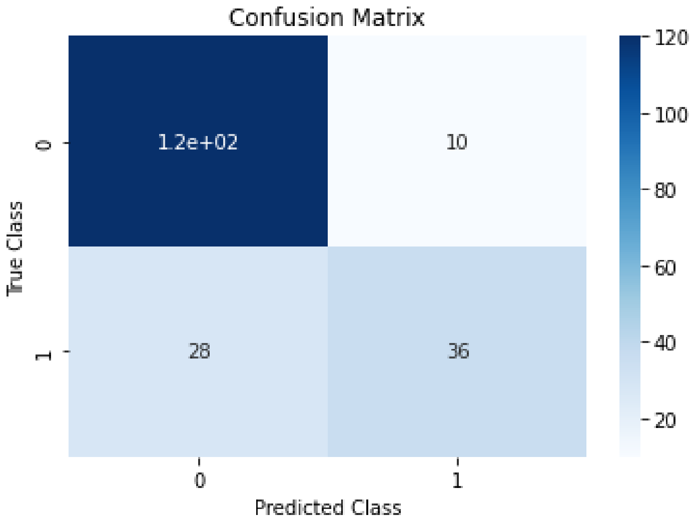
In [43]:

```
# generate a confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(cm)
```

```
Confusion matrix:
[[120  10]
 [ 28  36]]
```

In [45]:

```
# create a heatmap visualization of the confusion matrix
sns.heatmap(cm, annot=True, cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()
```



In [46]:

```
# generate a classification report
report = classification_report(y_test, y_pred)
print("Classification report:")
print(report)
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.81      0.92      0.86       130
           1       0.78      0.56      0.65        64

    accuracy                           0.80       194
   macro avg       0.80      0.74      0.76       194
weighted avg       0.80      0.80      0.79       194
```

In [47]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [48]:

```
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

Out[48]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [49]:

```
y_pred = clf.predict(X_test)
```

In [50]:

```
# calculate accuracy, precision, recall, and f1-score
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1))
```

```
Accuracy: 0.68
Precision: 0.52
Recall: 0.45
F1-score: 0.48
```

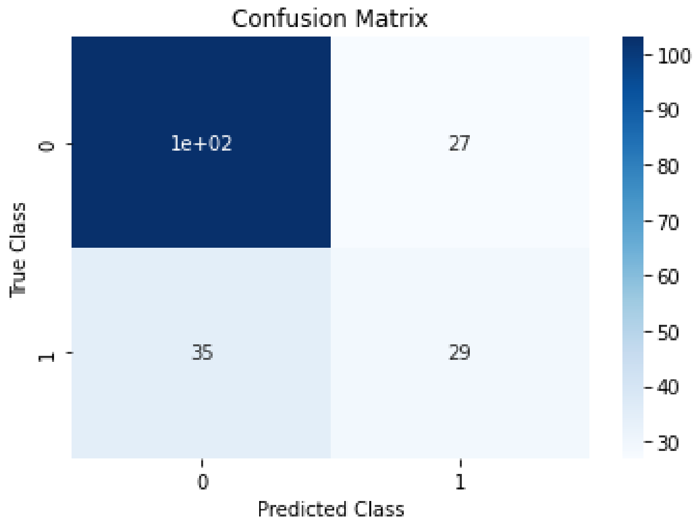
In [51]:

```
# generate a confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(cm)
```

```
Confusion matrix:
[[103  27]
 [ 35  29]]
```

In [52]:

```
# create a heatmap visualization of the confusion matrix
sns.heatmap(cm, annot=True, cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()
```



In [53]:

```
# generate a classification report
report = classification_report(y_test, y_pred)
print("Classification report:")
print(report)
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.75      0.79      0.77       130
           1       0.52      0.45      0.48        64

    accuracy                           0.68       194
   macro avg       0.63      0.62      0.63       194
weighted avg       0.67      0.68      0.67       194
```

In [54]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [55]:

```
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)
```

Out[55]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [56]:

```
y_pred = clf.predict(X_test)
```

In [57]:

```
# calculate accuracy, precision, recall, and f1-score
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1))
```

```
Accuracy: 0.75
Precision: 0.67
Recall: 0.48
F1-score: 0.56
```

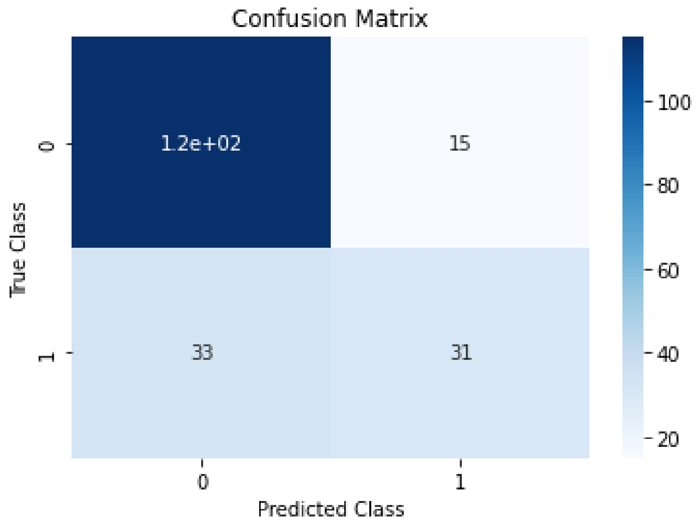
In [58]:

```
# generate a confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(cm)
```

```
Confusion matrix:
[[115  15]
 [ 33  31]]
```

In [60]:

```
# create a heatmap visualization of the confusion matrix
sns.heatmap(cm, annot=True, cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()
```



In [59]:

```
# generate a classification report
report = classification_report(y_test, y_pred)
print("Classification report:")
print(report)
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.78      0.88      0.83       130
           1       0.67      0.48      0.56        64

    accuracy                           0.75       194
   macro avg       0.73      0.68      0.70       194
weighted avg       0.74      0.75      0.74       194
```

In [61]:

```
from sklearn.svm import SVC
```

In [62]:

```
clf = SVC(kernel='linear')
clf.fit(X_train, y_train)
```

Out[62]:

```
▾        SVC

SVC(kernel='linear')
```

In [63]:

```
y_pred = clf.predict(X_test)
```

In [64]:

```
# calculate accuracy, precision, recall, and f1-score
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1))
```

```
Accuracy: 0.78
Precision: 0.76
Recall: 0.50
F1-score: 0.60
```

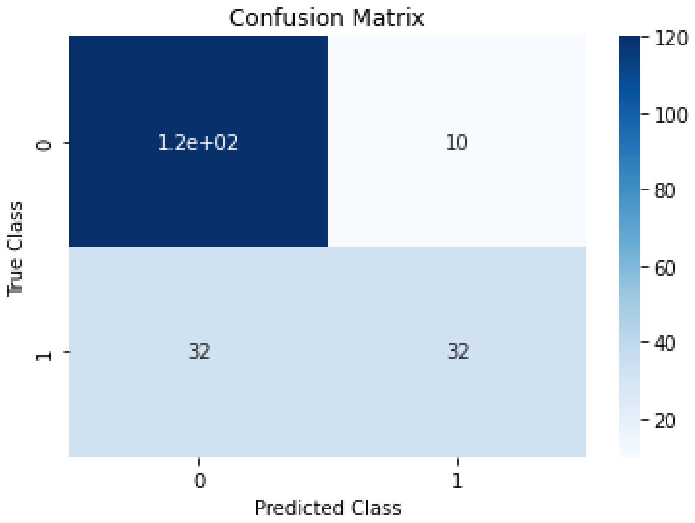
In [65]:

```
# generate a confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(cm)
```

```
Confusion matrix:
[[120  10]
 [ 32  32]]
```

In [66]:

```
# create a heatmap visualization of the confusion matrix
sns.heatmap(cm, annot=True, cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()
```


Confusion Matrix

In [67]:

```
# generate a classification report
report = classification_report(y_test, y_pred)
print("Classification report:")
print(report)
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.79      0.92      0.85       130
           1       0.76      0.50      0.60        64

    accuracy                           0.78       194
   macro avg       0.78      0.71      0.73       194
weighted avg       0.78      0.78      0.77       194
```

In [68]:

```
from sklearn.model_selection import GridSearchCV, train_test_split
```

In [69]:

```
param_grid = {'C': [0.1, 1, 10, 100], 'kernel': ['linear', 'rbf', 'poly'], 'degree': [2, 3, 4]
```

In [70]:

```
svc = SVC()
```

In [71]:

```
grid_search = GridSearchCV(estimator=svc, param_grid=param_grid, cv=5)
```

In [72]:

```
grid_search.fit(X_train, y_train)
```

Out[72]:

```
▸ GridSearchCV
▸ estimator: SVC
      ▸ SVC
```

In [73]:

```
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best parameters:", best_params)
print("Best score:", best_score)
```

```
Best parameters: {'C': 1, 'degree': 2, 'kernel': 'rbf'}
Best score: 0.7668515742128935
```

In [76]:

```
# Train Decision Tree classifier
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
dt_pred = dt.predict(X_test)
dt_acc = accuracy_score(y_test, dt_pred)

# Train Random Forest classifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
rf_acc = accuracy_score(y_test, rf_pred)

# Train SVM classifier
svm = SVC(C = 1, degree = 2, kernel = 'rbf')
svm.fit(X_train, y_train)
svm_pred = svm.predict(X_test)
svm_acc = accuracy_score(y_test, svm_pred)
```

In [77]:

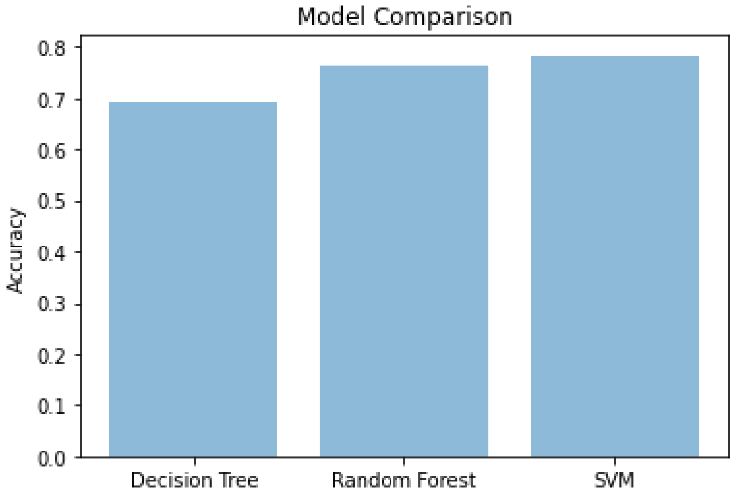
```
models = ['Decision Tree', 'Random Forest', 'SVM']
accuracies = [dt_acc, rf_acc, svm_acc]
```

In [78]:

```
x_pos = np.arange(len(models))
```

In [79]:

```
plt.bar(x_pos, accuracies, align='center', alpha=0.5)
plt.xticks(x_pos, models)
plt.ylabel('Accuracy')
plt.title('Model Comparison')
plt.show()
```


Model Comparison
Accuracy
Decision Tree
Random Forest
SVM